

Scalable Multiagent Coordination with Distributed Online Open Loop Planning

Lenz Belzner and Thomas Gabor

Institute for Informatics
Mobile and Distributed Systems
LMU Munich

Abstract. We propose distributed online open loop planning (DOOLP), a general framework for online multiagent coordination and decision making under uncertainty. DOOLP is based on online heuristic search in the space defined by a generative model of the domain dynamics, which is exploited by agents to simulate and evaluate the consequences of their potential choices. We also propose *distributed online Thompson sampling* (DOTS) as an effective instantiation of the DOOLP framework. DOTS models sequences of agent choices by concatenating a number of multiarmed bandits for each agent and uses Thompson sampling for dealing with action value uncertainty. The Bayesian approach underlying Thompson sampling allows to effectively model and estimate uncertainty about (a) own action values and (b) other agents' behavior. This approach yields a principled and statistically sound solution to the exploration-exploitation dilemma when exploring large search spaces with limited resources. We implemented DOTS in a smart factory case study with positive empirical results. We observed effective, robust and scalable planning and coordination capabilities even when only searching a fraction of the potential search space.

1 Introduction

We present a framework for efficient and scalable online multiagent coordination and decision making under uncertainty. We present distributed online open loop planning (DOOLP), a general framework based on online heuristic search in the space defined by a generative model of the domain dynamics, which is exploited by agents to simulate and evaluate the consequences of their potential choices.

We also present a particular instantiation of DOOLP, *distributed online Thompson sampling* (DOTS). DOTS models sequences of agent choices by concatenating a number of multiarmed bandits for each agent. In the distributed sequential setting, optimal bandit choices depend on subsequent action value estimates and preferences of other agents. We propose to use Thompson sampling as an efficient technique to estimate the value of actions by simulation. We show that the Bayesian approach underlying Thompson sampling allows to effectively model and estimate the uncertainty about (a) own action values and (b) other agents' behavior. DOTS uses probabilistic sampling strategies that are updated

in a Bayesian way based on simulated reward in order to solve the exploration-exploitation dilemma when exploring large search spaces with limited resources.

We implemented DOTS in a smart factory case study with more than 10^{22} states, and showed its applicability to distributed constraint optimization problems. We observed effective problem solving and coordination capabilities even when only searching a fraction of the potential search space. We compared different action selection strategies for dealing with the exploration-exploitation dilemma, and observed that Bayesian selection performed more effectively than other baseline approaches. We also evaluated the effect of gossip-like coordination when planning and observed that communication does not have to be global in order to maintain stable and effective global coordination results.

The remainder of the paper is structured as follows. Section 2 outlines related work. Section 3 describes the DOOLP framework for distributed online open loop planning. In Section 4 we discuss the DOTS algorithm as a Bayesian instantiation of DOOLP. We present an empirical case study in Section 5 and discuss our results. We conclude and outline venues for further research in Section 6.

2 Related Work

In this section, we recap decentralized Markov decision processes, online planning, open loop planning, multiarmed bandits and Thompson sampling.

2.1 Decentralized MDPs

A finite decentralized Markov decision process \mathcal{M} (DecMDP) is defined by a tuple $\mathcal{M} = (Ag, S, s_0, \{A_{ag}\}, P(S|S, A), R, h)$, where

- Ag is a set of agents,
- S is a set of states,
- $s_0 \in S$ is an initial state,
- A_{ag} is the set of actions of an agent $ag \in Ag$,
- $A = \otimes_{ag \in Ag} A_{ag}$ is the set of joint actions,
- $P(S|S, A)$ is the transition distribution, a probability distribution over (successor) states when executing a joint action in a given (preceding) state,
- $R : S \times A \times S \rightarrow \mathbb{R}$ is a reward function,
- $h \in \mathbb{N}^+$ is a planning horizon.

At every step $t < h$ in state $s_t \in S$, each agent $ag \in Ag$ executes an action $a_{ag} \in A_{ag}$, yielding the joint action $a_t = \bigcup_{ag \in Ag} a_{ag}$. The DecMDP then performs a transition w.r.t. the transition distribution, and an immediate reward $R(s_t, a_t, s_{t+1})$ is observed.

Solving a DecMDP consists in finding a policy $\Pi_{ag} : P(A_{ag}|S)$ for each agent $ag \in Ag$ such that some optimization criterion is satisfied when executing the set of policies. Note that joining the individual agents' policies yields a joint policy $\Pi : P(A|S)$.

A possible optimization criterion for finite DecMDPs is the expected cumulative reward. The cumulative reward CR of a DecMDP is defined as the sum of rewards observed when executing the policies of a set of agents.

$$\text{CR}(s_0, a_0, s_1, a_1, \dots, s_{h-1}, a_{h-1}, s_h) = \sum_{t=0}^{h-1} R(s_t, a_t, s_{t+1})$$

Optimizing the expected cumulative reward is then equal to maximizing the expectation of the cumulative reward.

$$\max \mathbb{E} \left[\sum_{t=0}^{h-1} R(s_t, a_t, s_{t+1}) \right]$$

We refer to [1] for an in-depth discussion of decentralized Markov decision processes.

2.2 Online Planning

Online planning, or local search, repeatedly interleaves a planning loop with an execution loop [2,3]. An online planning agent requires a probabilistic generative model of its domain dynamics, such as a transition distribution of a DecMDP. Online planning consists of two loops with different frequencies.

- A high-frequency planning loop samples a plan from a stochastic policy, simulates its consequences w.r.t. some optimization objective (e.g. the expected cumulative reward) and updates the policy in order to increase the probability of generating useful plans w.r.t. the given notion of utility.
- A low-frequency execution loop consists of sensing the current state of the environment, planning by repeating the inner loop multiple times, and executing the currently most viable action determined by the planning loop.

Online planning with its two loops is informally shown in Figure 1.

2.3 Open Loop Planning

Open loop planning is an approach to determine policies optimized w.r.t. some objective without storing information about the states that are intermediately encountered while simulating policy execution for evaluation purposes [4,2]. Given a set of actions A , we are only interested in finding a plan $p \in A^h$, and we are only keeping information about the action sequences in order to guide the planning process. That is, we reformulate the policy to $\Pi : P(A^h|S)$, now being a distribution of sequences of length h given a current state $s \in S$.

Open loop planning contrasts with closed loop planning, such as e.g. Monte Carlo Tree Search [5], where action selection is typically conditioned by the history of previously encountered states and executed actions.

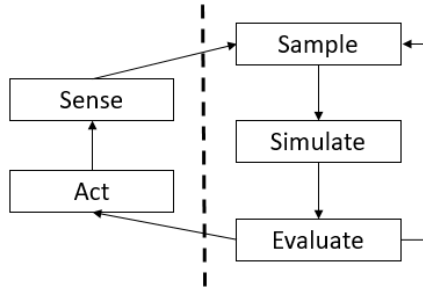


Fig. 1: Schematic representation of online planning. Note the two feedback loops of execution (left) and planning (right). The dashed vertical line indicates physical (left) and cognitive (right) components of an online planning agent.

2.4 Multiarmed Bandits

Multiarmed bandits (MAB) are a core framework for decision making. A bandit consists of a number of arms, each representing an agent’s choice. In our setting, each arm represents an action $a \in A$. Each arm provides a particular payoff, and the agent’s goal is to identify the most preferable arm. It can explore the bandit by pulling one arm at a time, and observe the corresponding payoff.

An MAB can be interpreted as a simple Markov decision process with a single state. In their basic formulation, MABs already provide a clear framework for studying the exploration-exploitation tradeoff inherent to decision making under uncertainty: Should the agent select the arm that previously showed to be most promising? Or should it go on exploring other options? For a recent survey of MAB and its variants, see [6].

2.5 Thompson Sampling

Thompson sampling (TS) is a Bayesian algorithm for solving an MAB. It was proposed decades ago [7], but only recently its astonishing effectiveness and generality have been identified [8,9,10].

TS infers a posterior distribution over p based on the observed arm payoffs and a prior assumption about the distribution of p . In general, the posterior is proportional to the likelihood of observed data D (i.e. an arm’s observed payoffs), multiplied by the prior distribution $P(\theta)$ over the parameters of interest, $\theta = p$ in our case (Equation 1).

$$P(\theta|D) \propto P(D|\theta)P(\theta) \quad (1)$$

TS maintains such a distribution $P(\theta)$ for each arm. The algorithm then samples a potential value for each arm from these distributions. It then plays the arm from whose distribution the maximum value has been sampled, observes the payoff, and uses this observation to update the corresponding distribution.

Repeating this process results in almost sure identification of the arm with the highest payoff. TS is schematically shown in Algorithm 1.

Algorithm 1 Thompson Sampling

- 1: **procedure** THOMPSON SAMPLING
 - 2: $\forall a \in A : \hat{v}_a \sim P_a(\theta)$
 - 3: play $\arg \max_a \hat{v}_a$ and observe result
 - 4: update $P_a(\theta)$ w.r.t. result
-

3 Distributed Online Open Loop Planning

In this Section, we describe the general DOOLP framework for multiagent coordination and decision making under uncertainty.

3.1 Overview

Distributed Online Open Loop Planning (DOOLP) realizes multiagent coordination by distributed simulation-based open loop planning. Each agent maintains a sampling policy that balances the exploitation-exploration tradeoff it faces when searching for an individual high-quality solution. Individual search is also influenced by other agents' current sampling strategies: Before simulating the consequences of a plan choice, an agent queries other agents it wants to coordinate with for a sample from their current policies. These queried plan samples are distributed w.r.t. the other agents' current preferences. As plan sampling, simulation and updating of sampling policies are performed iteratively, the other agents' samples influence the change of the individual policy by the degree of coordination that is present in expectation over the joint policy.

DOOLP is informally represented in Figure 2: Two coordinating online planning agents are shown. Their respective coordination by means of communication is highlighted in red. We emphasize that communication is part of the high-frequency planning loop of each agent.

3.2 DOOLP Formalization

We formalize the DOOLP framework as follows. Note that, while the formalization is closely related to DecMDPs (cf. Section 2), it is not necessary for DOOLP to specify a transition distribution and a reward function explicitly. Let the set of agents Ag , the state space S , the set of agent's actions $\{A_{ag}\}$, the set of joint actions $A = \otimes_{ag \in Ag} A_{ag}$ and the planning horizon h are defined as for a DecMPD (cf. Section 2). A DOOLP agent is a tuple $\mathcal{D} = (\Pi_{ag}, C, \Delta)$, with the following definitions.

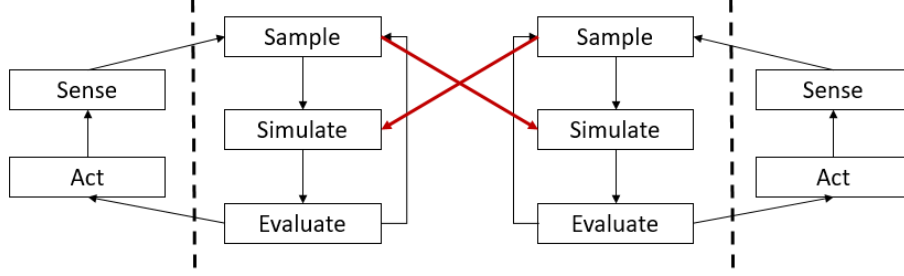


Fig. 2: Schematic representation of distributed online open loop planning. The dashed vertical lines separate physical and cognitive components of an online planning agent. Two coordinating agents are shown. Their respective coordination by means of communication is highlighted in red. Communication is part of the high-frequency planning loop of each agent.

- Each agent maintains a policy $\Pi_{ag} : P(A_{ag}^h | S)$. The policy is a probability distribution over plans (i.e. sequences of individual actions) given a state. This distribution is used to sample plans for which to simulate the consequences. The policy is updated after simulation in order to increase the probability of generating high-value plans.
- Each agent maintains a set of coordinating agents (e.g. neighbors, trusted agents, etc.) via a communication strategy $C : S \rightarrow 2^{A_g}$. At each sampling step, the joint plan to be simulated is constructed from (a) sampling from the own current policies and (b) querying all currently coordinating agents in $C(s)$ for a plan sample from their respective current individual policies.
- Each agent maintains a simulation of domain dynamics $\Delta : P(S \times \mathbb{R} | S \times A)$. This simulation is a probability distribution capturing aleatoric uncertainty of domain dynamics, e.g. stochastic action effects. Given a current state and a joint action it outputs a successor state and a reward associated with the generated transition. The reward is a signal about the quality of the observed transition w.r.t. an agent’s goals. DOOLP aims at maximizing the expected cumulative reward when executing a policy. Note that Δ mitigates the need to explicitly specify a DecMDP for a DOOLP agent. That is, the transition distribution and the reward function do not have to be explicitly formulated, but rather it is sufficient to provide a simulation of the application domain in order to apply DOOLP for distribution online coordination.

3.3 DOOLP Planning and Execution

Based on the DOOLP specification discussed above, we now describe the planning and execution loops for each agent.

Planning DOOLP’s inner high-frequency planning loop operates by repeatedly performing the following steps (cf. Algorithm 2).

- A planning agent samples an individual plan $p \in A^h$ from its own individual policy Π_{self} (line 2).
- The agent then queries all other agents in its set of coordinating agents $C(s)$ for samples from their current policies $\Pi_c, c \in C(s)$ (line 3). We emphasize that this communication in combination with a sensible exploration-exploitation strategy implicitly yields coordination of multiple agents. Note that in general $C(s)$ is unrestricted and may indeed contain all other present agents. For efficiency purposes, $C(s)$ may also be constructed by other means, e.g. based on topological neighborhood, trust values, etc.
- The agent simulates the joint plan built from its own plan and the coordinating agents' plans with the available mode of domain dynamics Δ and observes the associated rewards (lines 4 – 7).
- The agent computes the cumulative reward for each step in the simulation (i.e. the cumulative sum of future rewards from any given planning step) and updates its individual policy based on the cumulative rewards in a way that increases the probability of generating high-value plans (i.e. plans whose expected reward is high) (lines 8 – 11).

Algorithm 2 DOOLP planning loop

```

1: procedure PLAN( $s$ )
2:    $p \sim \Pi_{\text{self}}(s)$  ▷ sample plan from policy
3:    $p_{\text{joint}} \leftarrow p \cup \left( \bigcup_{c \in C(s)} p_c \right)$  ▷ query other agents' plans and join
4:    $r \leftarrow \text{nil}$  ▷ initialize plan rewards
5:   for  $a_i \in p_{\text{joint}}$  do
6:      $s, r_i \sim \Delta(s, a_i)$  ▷ simulate joint actions
7:      $r \leftarrow r :: r_i$  ▷ store reward
8:   for  $h - 1 \geq i \geq 0$  do
9:      $r_i \leftarrow r_i + r_{i+1}$  ▷ cumulative reward
10:  for  $a_i, r_i \in p, r$  do
11:    update  $\Pi_{\text{self}}$  w.r.t.  $(a_i, r_i)$  ▷ update individual policy

```

Execution DOOLP's outer execution loop repeatedly performs the following operations (cf. Algorithm 3):

- The current state is observed (line 3).
- The planning loop is executed until a user-defined event interrupts the loop, e.g. when a certain simulation budget has been reached, or an external event requires an agent's action (lines 4 and 5).
- The agent then uses the mode of its current policy Π_{self} to construct the plan with the highest expected future reward, and executes its first action (lines 6 and 7).

Algorithm 3 DOOLP execution loop

```
1: procedure DOOLP
2:   loop
3:     observe current state  $s$ 
4:     while not interrupted do
5:       PLAN( $s$ )
6:        $p \leftarrow \text{mode}(\Pi_{\text{self}})$ 
7:       execute  $p_0$ 
```

We assume communication of a DOOLP agent (listening and answering queries) to run in parallel to the execution loop. We assume this communication routine to be non-blocking, and to be able to access the current state and the policy of the agent in order to sample and return a currently viable plan to any querying agent.

4 Distributed Online Thompson Sampling

DOOLP is instantiated by implementing the following operations:

1. Representation of the policy Π_{self} .
2. Sampling plans from the policy.
3. Updating the policy given observed action-reward tuples.

In the following, we describe each of these points for distributed online Thompson sampling as a DOOLP instance.

4.1 Policy Representation

DOTS uses a stack of multiarmed bandits to represent the policy $\Pi_{ag} : P(A_{ag}^h | S)$ of an agent $ag \in Ag$. For each planned step, DOTS maintains a bandit that models an agent’s belief about action values for the corresponding step in the plan generated by the policy, based on previous simulations of plans via Δ .

Action Values By simulating execution of an action sequence from an initial state with Δ , an agent obtains a sequence of states, actions and rewards $s_1, a_1, r_1, \dots, s_h, a_h, r_h$. Given such data, we define the value of an action $a_i, 1 \leq i \leq h$ as the cumulative reward gained onwards from executing that action.

$$\hat{V}(a_i) = \sum_{i \leq j \leq h} r_j$$

In order to estimate an action’s expected value, DOTS maintains a buffer of observed action sample values $X_{a,i}$ for each action $a \in A$ at each step $i \leq h$. As multiagent coordination yields a moving target distribution of values (i.e. concept drift occurs in the process of coordination), we maintain the buffer in a sliding window fashion.

Bayesian Estimation of Action Value Expectation Given a buffer of observed action values $X_{a,i}$ for an action a at depth i , an agent can estimate the corresponding action’s value distribution in a Bayesian way as described in the following. DOTS assumes that the expectation of an action’s value is normally distributed with mean μ and precision τ .

$$x \sim \mathcal{N}(\mu, \tau^{-1}) \quad (2)$$

These parameters are unknown initially, and are to be estimated by DOTS based on simulation of action sequences with Δ . We place a normal-gamma prior over the parameters μ and τ of this distribution to model an agent’s initial uncertainty about μ and τ . The normal-gamma prior is parametrized by a prior mean μ_0 , the number of prior mean pseudo-observations λ_0 , the number of prior variance pseudo-observations α_0 and β_0 , such that $\frac{\beta_0}{\alpha_0}$ is the prior’s sample variance. Given these parameters, we can sample prior mean and precision from the corresponding normal-gamma distribution.

$$(\mu, \tau) \sim \mathcal{NG}(\mu_0, \lambda_0, \alpha_0, \beta_0) \quad (3)$$

The normal-gamma posterior parameters of the action value distribution μ and τ after observing action values $X = x_1, \dots, x_n$ are computed as follows.

$$P(\mu, \tau | X) = \mathcal{NG} \left(\frac{\lambda_0 \mu_0 + n \bar{x}}{\lambda_0 + n}, \lambda_0 + n, \alpha_0 + \frac{n}{2}, \beta_0 + \frac{1}{2} \left(ns + \frac{\lambda_0 n (\bar{x} - \mu_0)^2}{\lambda_0 + n} \right) \right) \quad (4)$$

Here, $n = |X|$ is the number of observations, $\bar{x} = \frac{1}{n} \sum_{x_i \in X} x_i$ is the observations’ mean, and $s = \frac{1}{n} \sum_{x_i \in X} (x_i - \bar{x})^2$ is the observations’ sample variance.

4.2 Sampling from the Policy

Sampling a plan from the policy Π_{self} (cf. Algorithm 2, line 2) is performed via Thompson sampling (cf. Section 2.5). Let $h \in \mathbb{N}^+$ be the planning depth. For each planning step $i \in \{0, \dots, h\}$ and each action $a \in A$, the agent has observed the rewards-to-go $X_{a,i}$. To sample a plan from Π_{self} an agent builds a normal-gamma distribution $P(\mu_{a,i}, \tau_{a,i})$ from the corresponding previously observed $X_{a,i}$ as defined by Equation 4. It then samples a parametrization of the reward distribution for each planning step $i \in \{0, \dots, h\}$ and each action $a \in A$. The sequence of actions maximizing $\mu_{a,i}$ for each i then form the sampled plan. Algorithm 4 shows the corresponding sampling procedure.

4.3 Updating the Policy

Updating the policy is implicitly done by updating the set of observed action values $X_{a,i}$ for actions that are simulated in the planning loop (cf. Algorithm 2). Changing $X_{a,i}$ directly influences the distributions $P(\mu_{a,i}, \tau_{a,i} | X_{a,i})$, which in turn define the density of sampled plans (cf. Algorithm 4, line 3).

Algorithm 4 Sampling a plan from the policy $\Pi_{\text{self}} = P(\cdot|X_{a,i})$ with Thompson sampling, given observed rewards-to-go $X_{a,i}$. For the DOTS algorithm, this procedure implements line 2 in Algorithm 2 of the DOOLP framework. It is also used to sample plans for answering communication queries from other agents.

```

1:  $p \leftarrow \text{nil}$ 
2: for  $0 \leq i < h$  do
3:    $\forall a \in A_{\text{self}} : (\mu_{a,i}, \tau_{a,i}) \sim P(\cdot|X_{a,i})$ 
4:    $a_i \leftarrow \arg \max_a \mu_{a,i}$ 
5:    $p \leftarrow p :: a_i$ 

```

As multiagent planning yields a moving target (due to agents concurrently changing/adapting their preferences), we treat the $X_{a,i}$ in a sliding window fashion in order to only reflect the most recent observed/simulated action values for a particular action choice. That is, all $X_{a,i}$ are implemented as buffers in a first-in, first-out manner, only keeping track of the most recent evaluations.

5 Case Study

We empirically evaluated the effectiveness of DOTS in a smart factory case study.

5.1 Domain Setup

We considered a setting where various items are to be processed in a smart factory consisting of a number of different machines. Each item carries constraints on the type of processing that it has to pass in order to proceed, and also on the order of the processing steps. Each machine is associated with a processing type, a processing cost and a processing failure probability. The latter is exemplary for domain inherent stochasticity.

When not enqueued at a machine, agents decide at which machine to enqueue in order to get their processing steps done as fast as possible. This implicitly requires coordination of requests and resources, as a machine can only process one item at a time. Agents also have the option to do nothing, i.e. to wait. The reward generated by the domain depends on the number of processed request, as well as the processing cost associated with the processing machines.

Note that the resulting problem grows exponentially with the number of agents. For a setting with i items and m machines, and plans of length $|p|$, the resulting search space is of cardinality $(m+1)^{i|p|}$. This means that in a setting with 8 items, 4 machines and a plan length of four there are more than 10^{22} joint plan options for the items in any given situation. Also, as machines have a Bernoulli failure probability, there is a very high branching factor regarding the consequences of joint actions.

To ease reproducibility, an implementation of our experimental setup can be downloaded from <https://github.com/jazzbob/doolp>.

5.2 Coordination Variants

We evaluated DOTS in our setup by comparing it to three baselines.

- As a first baseline, we used a distributed random search approach, that we label Vanilla Monte Carlo (VMC). For VMC, each agent samples potential plans uniformly at random. There is no sampling policy update step of the policy due to observed simulation rewards. The joint sampled plans are then simulated and each agent keeps track of the plan that achieved the best joint value so far. The first action of the best found plan is executed when the planning loop is interrupted, and the process repeats. Note that VMC does not update its sampling policy based on observed action-reward tuples. Applying VMC in a multiagent scenario can therefore be interpreted as implicit coordination. I.e., with VMC coordination only occurs due to the resulting states the system encounters at runtime, but not due to a dedicated coordination effort.
- As a second baseline, we used an instantiation of the DOOLP framework where sampling is performed in an ϵ -greedy manner. I.e., with probability $1 - \epsilon$, the action with maximum mean previously observed rewards-to-go is selected. With probability ϵ , a random action is sampled uniformly from the action set for exploration purposes. We set ϵ to 0.1 in our experiments. This DOOLP instantiation updates the sampling policy by building the mean observed cumulative reward for each action at each planning step. In contrast to the VMC baseline, this baseline performs explicit coordination as specified by DOOLP. However, in contrast to DOTS, it does not model uncertainty about its cumulative reward estimates, but rather uses a maximum likelihood approach for value estimation.
- As a third baseline, we used an instantiation of the DOOLP framework where sampling is performed with the UCB algorithm [11]. UCB is a well-known selection strategy for multiarmed bandits based on upper confidence bounds of the reward expectation for each arm, yielding an exploration behavior known as *optimism in the face of uncertainty* [4]. Let n be the number of a multiarmed bandit has been sampled, and let n_a be the number of samples for a particular action a . The UCB score for an action a is given by the following term.

$$\text{UCB}_a = \bar{x}_a + c \cdot \sqrt{\frac{2 \ln n}{n_a}}$$

Here, \bar{x}_a is the sample mean of observed rewards for action a and $c > 0$ is a constant weighting the exploration term. We set $c = 1$ in our experiments. UCB selects the action that maximized the UCB score, i.e. $\max_{a \in A} \text{UCB}_a$. Policy updating is done by keeping track of observed rewards and the number of samples for each action, directly influencing the UCB scores.

We evaluated DOTS in various settings, with similar results. Here we report on a setting with 8 items, 4 machines and a planning depth of 4. For each executed action a number $n \in \{64, 128, 256, 512\}$ of simulations were performed

(i.e. planning loop interruption occurred after n simulations). Note that in all cases, $n \ll 10^{22}$, that is, only a fraction of the joint plan space is searched. We used a sliding window size of 10 for the size of the rewards-to-go buffers $X_{a,i}$.

We used the following normal-gamma prior to model initial agent uncertainty about action value distributions (cf. Section 4.1).

$$\mu_0 = 0, \lambda_0 = 1, \alpha_0 = 1, \beta_0 = 100$$

Note that the influence of prior parameters on the posterior distribution is reduced with increasing numbers of observations used for inferring the posterior.

5.3 Results

We posed the following research questions to be answered by our experiments.

1. Is DOOLP realizing coordination effectively by interleaving planning, communication and execution?
2. Does DOTS' Bayesian uncertainty treatment yield a positive effect on coordination quality?
3. Is DOOLP robust w.r.t. communication coverage, i.e. is it scalable to many communicating agents?

Coordination Effectiveness Figure 3 shows the average cumulative scores w.r.t. discrete time steps (i.e. number of execution loops) achieved by the agents planning their actions in a distributed manner with DOTS. Also shown are results of the three baseline approaches (VMC, ϵ -greedy, UCB). The shaded areas show one standard deviation of the results. DOTS consequently outperformed the baseline approaches regardless of the number of simulations and policy updates performed before executing agents' actions. The ϵ -greedy variant of DOOLP was able to reach the performance of DOTS for 512 planning iterations. UCB performed weaker than both in all settings. As VMC yields significantly lower rewards in all settings than all DOOLP variants, we conclude that explicit coordination is indeed realized by instantiations of DOOLP. Given our observations, we give a positive answer to questions 1 and 2 above.

Coordination Robustness and Scalability We evaluated the robustness and scalability of DOTS w.r.t. to communication coverage. These are important characteristics of multiagent planning, as communication uses possibly scarce or expensive resources (e.g. bandwidth), and is typically prone to failure (e.g. message loss). For each iteration of the planning loop, we randomly dropped a fraction of agents from the communication set C , resulting in gossip communication between coordinating agents [12]. This also reduces the computational resources needed for simulation, as only the remaining agents in C were participating in queried joint plans.

Figure 4 shows the corresponding comparison of average achieved cumulative scores for different communication drop rates (i.e. fractions of agents not included

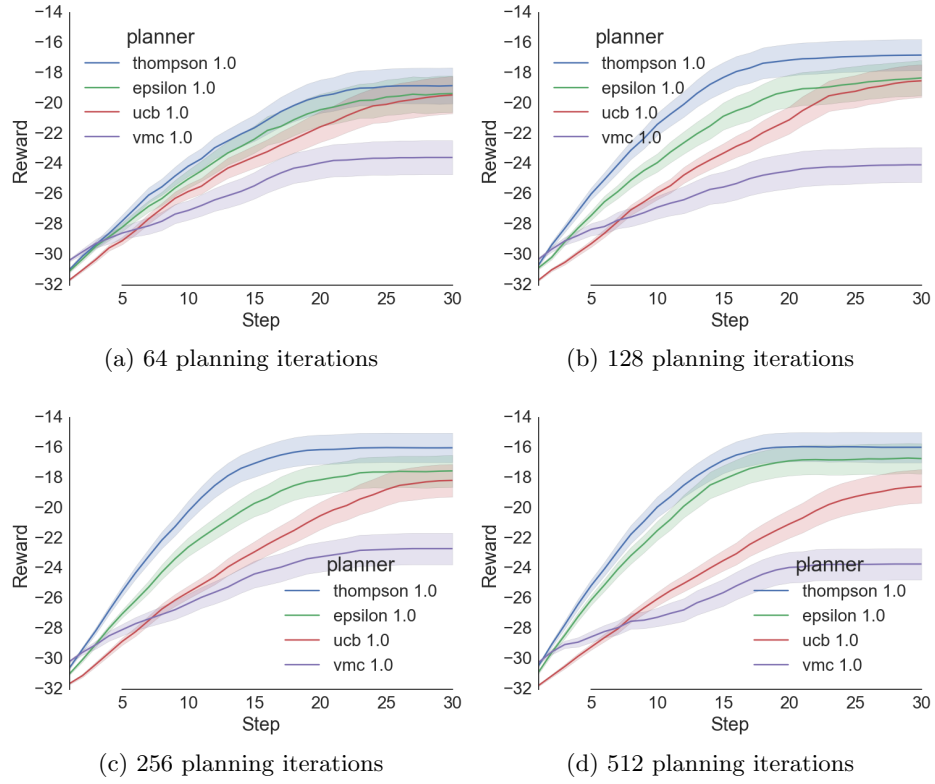


Fig. 3: Average achieved rewards (50 runs) over time for different planner variants and planning iterations per executed action. Shaded areas show one standard deviation. Best viewed in color.

in the joint plans that are simulated) for 512 planning iterations. We measured drop rates of 0, 0.25, 0.5 and 0.75. We observe that DOTS and ϵ -greedy DOOLP are robust against limiting communication between agents. Interestingly, the UCB variant increased its performance when reducing communication. We conjecture that this is due to the optimistic nature of UCB sampling. It is known that early optimism may result in poor local optima in multiagent coordination [13]. UCB sampling may emphasize this effect, and reducing communication possibly mitigates it. As expected, VMC does not suffer significantly from reduced communication as agents guess their individual plans regardless of communication (i.e. there is no sampling strategy update due to communication). We observed similar results for fewer planning iterations, with DOTS becoming slightly sensitive to communication drop rates, still yielding the best results of all compared DOOLP variants in all cases (see Appendix A for results). Given our results, we give a positive answer to question 3 above.

We conclude that explicit coordination with the DOOLP framework provides a scalable and efficient way for multiagent coordination under uncertainty, and that DOTS’ Bayesian modeling of action value uncertainty yields additional coordination performance in comparison to maximum likelihood estimation as done by the ϵ -greedy baseline and action selection based confidence bounds as done by the UCB variant.

6 Conclusion & Further Work

We proposed *distributed online open loop planning* (DOOLP), a framework for scalable online multiagent coordination and decision making under uncertainty. We also proposed a particular instantiation of DOOLP, *distributed online Thompson sampling* (DOTS). DOTS uses a Bayesian approach for modeling uncertainty in order to achieve coordination in cooperative multiagent settings. We have presented a formal description of DOOLP and DOTS, and evaluated its effectiveness empirically on a smart factory case study. We also evaluated the robustness of various DOOLP variants w.r.t. communication rates between agents, and observed a highly robust coordination quality w.r.t. communication rates between coordinating agents. Our results show that DOTS is a viable candidate for robust and scalable online multiagent coordination under uncertainty.

The DOOLP framework could straightforwardly be extended to more complex settings, including asynchronous coordination, local agent knowledge, heterogeneous reward functions, actions duration planning [14] or different optimization objectives such as risk metrics or quality of service (see e.g. [15,16]).

Another direction would be to incorporate simulation models learned from data (e.g. runtime observations), and to deal with model uncertainty arising from the learning process in the planning process. Also, statistical system verification under these constraints is a direction of current research. See e.g. [17] for recent work of the authors in this direction.

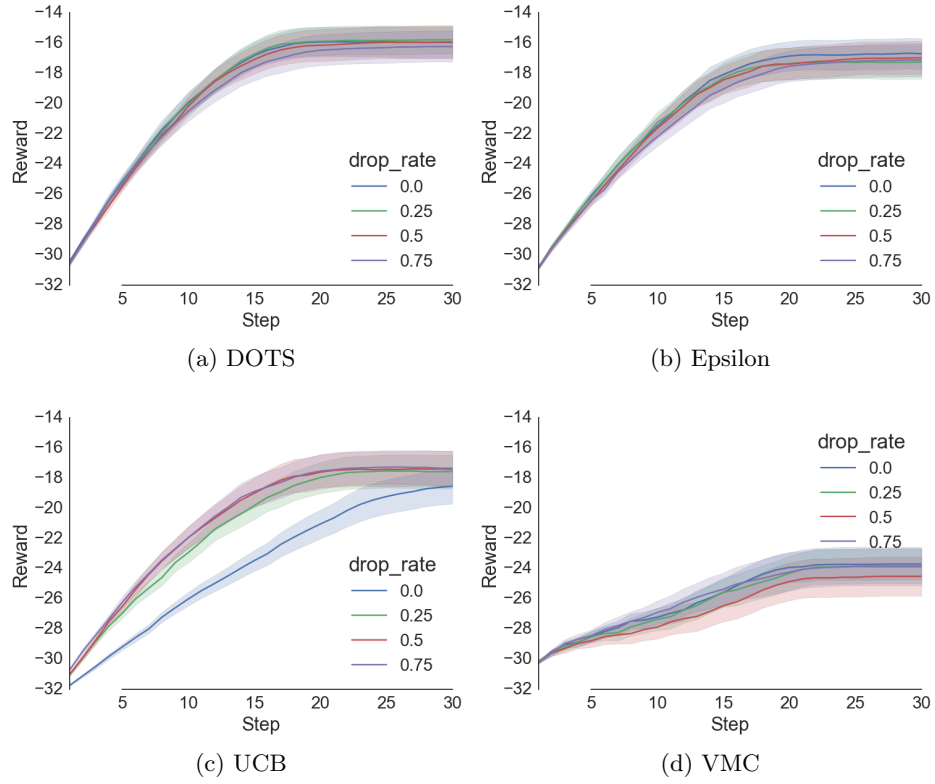


Fig. 4: Average achieved rewards (50 runs) over time for different planner variants and communication drop rates. Shaded areas show one standard deviation. 512 planning iterations per executed action. Best viewed in color.

Another interesting venue for future research is to integrate global, emergent phenomena into the local planning and coordination efforts. This could for example be achieved by learning predictive models (e.g. [18,19]) about global effects of interest (e.g. safety or quality w.r.t. global system requirements), emerging from individual interaction of all agents. The learned predictive model could in turn be used to guide the local planning processes [20,21].

References

1. Oliehoek, F.A., Amato, C.: A concise introduction to decentralized POMDPs. Springer (2016)
2. Weinstein, A., Littman, M.L.: Open-loop planning in large-scale stochastic domains. In: AAAI. (2013)
3. Belzner, L., Hennicker, R., Wirsing, M.: Onplan: A framework for simulation-based online planning. In: International Workshop on Formal Aspects of Component Software, Springer (2015) 1–30
4. Bubeck, S., Munos, R.: Open loop optimistic planning. In: COLT. (2010) 477–489
5. Chaslot, G.: Monte-carlo tree search. Maastricht: Universiteit Maastricht (2010)
6. Kuleshov, V., Precup, D.: Algorithms for multi-armed bandit problems. arXiv preprint arXiv:1402.6028 (2014)
7. Thompson, W.R.: On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* **25**(3/4) (1933) 285–294
8. Ortega, P.A., Braun, D.A.: A bayesian rule for adaptive control based on causal interventions. arXiv preprint arXiv:0911.5104 (2009)
9. Chapelle, O., Li, L.: An empirical evaluation of thompson sampling. In: Advances in neural information processing systems. (2011) 2249–2257
10. Kaufmann, E., Korda, N., Munos, R.: Thompson sampling: An asymptotically optimal finite-time analysis. In: International Conference on Algorithmic Learning Theory, Springer (2012) 199–213
11. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. *Machine learning* **47**(2-3) (2002) 235–256
12. Shah, D., et al.: Gossip algorithms. *Foundations and Trends® in Networking* **3**(1) (2009) 1–125
13. Panait, L., Tuyls, K., Luke, S.: Theoretical advantages of lenient learners: An evolutionary game theoretic perspective. *Journal of Machine Learning Research* **9**(Mar) (2008) 423–457
14. Belzner, L.: Time-adaptive cross entropy planning. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, ACM (2016) 254–259
15. Rockafellar, R.T., Uryasev, S.: Optimization of conditional value-at-risk. *Journal of risk* **2** (2000) 21–42
16. Belzner, L., Gabor, T.: Qos-aware multi-armed bandits. In: Foundations and Applications of Self* Systems, IEEE International Workshops on, IEEE (2016) 118–119
17. Belzner, L., Gabor, T.: Bayesian verification under model uncertainty. In: SeSCPS @ ICSE 2017. (to appear)
18. Schmidhuber, J.: Deep learning in neural networks: An overview. *Neural networks* **61** (2015) 85–117
19. Bengio, Y., Goodfellow, I.J., Courville, A.: Deep learning. *Nature* **521** (2015) 436–444

20. Hester, T., Stone, P.: Texple: real-time sample-efficient reinforcement learning for robots. *Machine learning* **90**(3) (2013) 385–429
21. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. *Nature* **529**(7587) (2016) 484–489

A Coordination Robustness and Scalability for Different Numbers of Planning Iterations

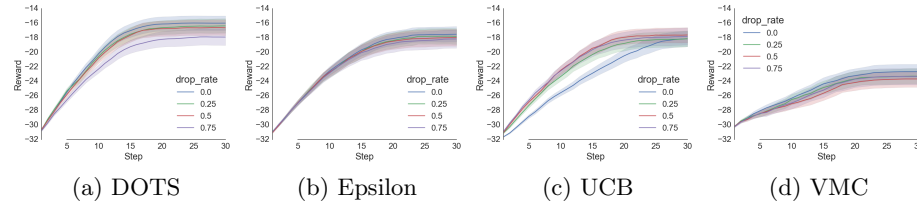


Fig. 5: Average achieved rewards (50 runs) over time for different planner variants and communication drop rates. Shaded areas show one standard deviation. 256 planning iterations per executed action. Best viewed in color.

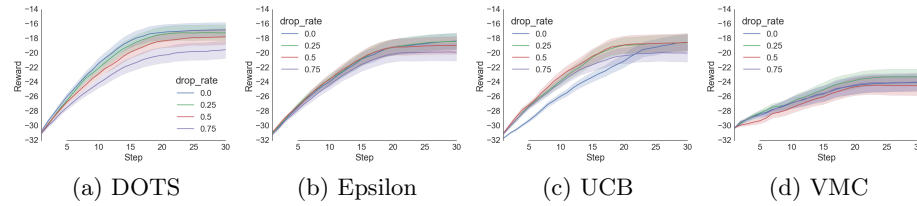


Fig. 6: Average achieved rewards (50 runs) over time for different planner variants and communication drop rates. Shaded areas show one standard deviation. 128 planning iterations per executed action. Best viewed in color.